苏 州 大 学 实 验 报 告

院、系	计算机学院	年级专业	软件工程	姓名	朱金涛	学号	2327406014
课程名称	机器学习综合实践					成绩	
指导教师	李俊涛	同组实验者	无		实验日期	2025 年 11 月 5 日	

实 验 名 称　基于 SVM 的验证码识别

一.实验目的

完成一个基于 SVM（支持向量机）的验证码识别任务，并且要求对比和分析不同核函数对验证码识别效果所造成的影响 。

二.实验内容

使用支持向量机（SVM）技术来完成一个验证码识别任务。

本次实验的核心内容主要分为两个部分：

1. 获取/构造验证码图片信息，并做**切割、打标、特征**提取等预先操作。

2. 实现 SVM 验证码识别： 学生需要利用 SVM 算法，构建一个能够识别验证码的模型。

3. 对比核函数： 在完成识别任务的基础上，需要**重点对比 SVM 中不同核函数**（Kernel Functions）对最终识别效果的影响 。

三.实验步骤和结果

（一） 数据采集

链接教程中所提供的网页现已不存在。若按照其方法，所提取的.png 文件全是 HTML 格式的文字段，显示网页不存在。考虑到本次实验重点不在于数据集采集，而在于对比 SVM 的不同核函数的效果对比，所以我采用了 **Pillow** 库来自动生成图像的方法，代替从真实世界的网站上爬取。这是一种 **“程序化图像生成”** (Procedural Image Generation)。它是一个“合成数据集”，完全由代码（主要是 Pillow 库）和**随机数**（random 库）创造出来。

生成代码如下（包括随机噪点、旋转、干扰线的设置）：

```python
# 3a. 添加随机化
        font = random.choice(fonts)
        x_offset = random.randint(-2, 2)
        y_offset = random.randint(-2, 2)
        angle = random.randint(-20, 20) # 旋转

        # 3b. 绘制带旋转的字符
        char_canvas_size = (int(font_size * 1.5), int(font_size * 1.5))
        char_img = Image.new('RGBA', char_canvas_size, (255, 255, 255, 0))
        char_draw = ImageDraw.Draw(char_img)
        char_draw.text((2, 0), char, fill=(0, 0, 0, 255), font=font)

        rotated_char = char_img.rotate(angle, resample=Image.Resampling.BICUBIC, expand=False)

        paste_x = START_X + j * (CHAR_WIDTH + CHAR_SPACING) + x_offset
        paste_y = START_Y + y_offset

        img.paste(rotated_char, (paste_x, paste_y), rotated_char)

    # 4. 转换回 'L' (灰度)
    final_img = img.convert('L')
    final_draw = ImageDraw.Draw(final_img)

    # 5. 增加噪点
    for k in range(25):
        nx = random.randint(0, IMG_WIDTH - 1)
        ny = random.randint(0, IMG_HEIGHT - 1)
        final_draw.point((nx, ny), fill=0)

    # 6. 增加干扰线
    line_x1 = random.randint(0, IMG_WIDTH // 2)
    line_y1 = random.randint(0, IMG_HEIGHT)
    line_x2 = random.randint(IMG_WIDTH // 2, IMG_WIDTH)
    line_y2 = random.randint(0, IMG_HEIGHT)
    final_draw.line((line_x1, line_y1, line_x2, line_y2), fill=0, width=1)
```

生成的效果图片如下：

（二） 图像处理与切割

1. 图像预处理：

首先定义了一个“二值化”函数。当 process_and_label_images 函数运行时，它会遍历 medium_images 文件夹中的每张图片，将它们从“灰度图”（带噪点）转换成“纯黑白”图片，以便于机器分析。

代码实现：

```python
def get_bin_table(threshold=140):
    """
    (来自教程) 获取灰度转二值的映射table
    """
    table = []
    for i in range(256):
        if i < threshold:
            table.append(0)
        else:
            table.append(1)
    return table
```

2. 图像切割：

按照教程中的“固定坐标切割”方法。所有数字都在固定的位置（x=2, x=15, x=28, x=41）。将每张 52x18 的黑白验证码，切割成 4 张 8x18 的“小图片”。

代码实现：

```python
def get_crop_imgs(img_binary):
    """
    “固定坐标”切割
    """
    child_img_list = []
    for i in range(4):
        x = 2 + i * (8 + 5)
        y = 0
        child_img = img_binary.crop((x, y, x + 8, y + 18))
        child_img_list.append(child_img)
    return child_img_list
```

3. 处理与打标

相比于教程中爬取现实世界网站验证码图片的方法，利用 Pillow 库在生成的时候其实就已经知道了图片中具体数字是什么，因此可以直接让每一个图片文件以其数字命名，例如：**0136_557.png**。

这样一来其实可以省去**“人工分类”**这一繁琐且相对来说浪费时间的操作（因为只是简单的将数字分类到不同文件夹）

具体切割操作如下：

当处理一张名为 3852_0.png 的图片时，**首先**把图片切割成 4 张小图，**随后**从文件名中读取标签"3852"，把第 1 张小图（切 x=2 位置的图），保存到 training_dataset/3/文件夹下；**之后**把第 2 张小图（盲切 x=15 位置的图），保存到 training_dataset/8/文件夹下...以此类推。

代码实现：

```python
def process_and_label_images():
    print(f"\n--- 步骤二：开始处理图片并自动打标 ---")

    GENERATED_IMG_FOLDER = 'medium_images'
    LABELED_DATA_FOLDER = 'training_dataset'

    if not os.path.exists(GENERATED_IMG_FOLDER):
        print(f"错误：找不到 {GENERATED_IMG_FOLDER}。")
        return

    if os.path.exists(LABELED_DATA_FOLDER):
        shutil.rmtree(LABELED_DATA_FOLDER)
    os.makedirs(LABELED_DATA_FOLDER)

    binary_table = get_bin_table()
    total_images = 0

    for img_name in os.listdir(GENERATED_IMG_FOLDER):
        if not img_name.endswith('.png'):
            continue

        label = img_name.split('_')[0]
        img_path = os.path.join(GENERATED_IMG_FOLDER, img_name)
        img = Image.open(img_path)
        img_gray = img.convert('L')
        img_binary = img_gray.point(binary_table, '1')

        child_images = get_crop_imgs(img_binary)

        if len(child_images) != len(label):
            continue

        for i in range(len(label)):
            char_label = label[i]
            child_img = child_images[i]
            char_folder = os.path.join(LABELED_DATA_FOLDER, char_label)
            if not os.path.exists(char_folder):
                os.makedirs(char_folder)
            save_name = f"{img_name.split('.')[0]}_{i}.png"
            child_img.save(os.path.join(char_folder, save_name))
            total_images += 1

    print(f"--- 步骤二：完成！共 {total_images} 张切割好的图片已存入 {LABELED_DATA_FOLDER} ---")
```

（三） 特征提取

1. 特征提取（get_feature）

将“步骤二”切割好的单张数字图片（img）转换成 SVM 能理解的形式，即特征向量。

具体方法和教程一致，通过“行列求和”来生成：

a) **尺寸归一化：** 首先使用 img.resize((8, 18)) 将所有（可能大小不一的）输入图片强制统一为 8x18 像素的标准“模板”。

b) **横向特征：**它逐行（height=18 行）扫描，统计每一行有多少个黑色像素（pix_cnt_x），并将这 18 个数字存入列表。

c) **纵向特征：**它再逐列（width=8 列）扫描，统计每一列有多少个黑色像素（pix_cnt_y），并将这 8 个数字追加到同一个列表中。

d) **返回：** 它最终返回一个包含 18 + 8 = 26 个数字的列表（pixel_cnt_list），这个列表就是这张图片的“特征向量”

代码实现：

```python
def get_feature(img):
    try:
        # 1. 尺寸归一化 (Normalization)
        img = img.resize((8, 18))
    except Exception as e:
        return None

    width, height = img.size
    pixel_cnt_list = []

    # 2. 横向特征 (18个)
    for y in range(height): # 18
        pix_cnt_x = 0
        for x in range(width): # 8
            if img.getpixel((x, y)) == 0:
                pix_cnt_x += 1
        pixel_cnt_list.append(pix_cnt_x)

    # 3. 纵向特征 (8个)
    for x in range(width): # 8
        pix_cnt_y = 0
        for y in range(height): # 18
            if img.getpixel((x, y)) == 0:
                pix_cnt_y += 1
        pixel_cnt_list.append(pix_cnt_y)

    # 4. 返回 26 维的特征向量
    return pixel_cnt_list
```

2. 数据集生成与格式化

这是“步骤三”的核心。它负责调用 get_feature，并将成千上万个特征向量格式化为 libsvm 库能读取的.txt 文件。

a) **遍历文件夹：** 循环 range(10)，分别进入 training_dataset/0、training_dataset/1...training_dataset/9 这些在“步骤二”中自动生成的文件夹。

b) **分割数据：** 将每个文件夹中的图片列表按 80/20 的比例分割，split_index 之前的图片用于训练（train_feature.txt），之后的用于测试（test_feature.txt）。

c) **调用特征提取：** 它在循环中打开每一张小图片（im = Image.open(im_path)），并调用 get_feature(im)将其转换为一个 26 维的特征向量。

d) **格式化：** 这是最关键的一步。它将特征向量转换成 libsvm 的标准格式：[标签] [索引 1]:[值 1] [索引 2]:[值 2] ...

- **例如，一个标签为'3'，特征为[5, 2, ...]的图片，会被格式化为字符串："3 1:5 2:2 ... 26:x"。**

e) **写入文件：** 它将格式化后的字符串行（line）根据 80/20 的分割，分别写入 train_feature.txt 或 test_feature.txt。

代码实现：

```python
def create_feature_files():
    # 1. 遍历 0-9 十个文件夹
    for n in range(10):
        char_label = str(n)
        char_folder = os.path.join(LABELED_DATA_FOLDER, char_label)
        if not os.path.exists(char_folder):
            continue
        img_list = os.listdir(char_folder)

        # 2. 按 80/20 分割训练集和测试集
        split_index = int(len(img_list) * 0.8)

        for i, im_file in enumerate(img_list):
            im_path = os.path.join(char_folder, im_file)
            try:
                im = Image.open(im_path)
            except:
                continue

            # 3. 调用特征提取
            im_feature = get_feature(im)
            if im_feature is None:
                continue

            # 4. 格式化为 libsvm 标准字符串
            line_parts = [char_label]
            for z_index, z_value in enumerate(im_feature):
                line_parts.append(f"{z_index + 1}:{z_value}")
            line = ' '.join(line_parts) + '\n'

            # 5. 写入对应的 .txt 文件
            if i < split_index:
                f_train.write(line)
                total_train += 1
            else:
                f_test.write(line)
                total_test += 1
```

（四）模型训练与测试

1. 导入 LibSVM 库

在导入过成功遇到了一个版本性问题：对于 libsvm (旧版包)而言，直接 import svmutil 即可找到对应的库。教程源码中也是用的旧包，我跟着做的时候一直显示未找到 svmutil 包，前前后后共考虑了：包文件缺失、下载到其他内核、jupyter 异常、IDE 异常等情况，但都不是问题本源所在。最后在 CSDN 论坛中找到了答案，由于我是直接 pip 下载了 libsvm-official (pip 新包)，要想调用这个包需要用 import libsvm.svmutil 的形式。

2. 加载特征数据

这段代码是训练的“燃料”。使用 libsvm 库自带的 svm_read_problem 函数，读取在“步骤三”中生成的.txt 特征文件，并将它们加载到内存中，区分为“标签”（y_train）和“特征数据”（x_train）。

代码实现：

```
try:
        # y_train是标签列表, x_train是特征数据列表
        y_train, x_train = svm_read_problem(TRAIN_FILE)
        y_test, x_test = svm_read_problem(TEST_FILE)
    except Exception as e:
        print(f"错误：读取特征文件失败。{e}")
        return
```

3. 循环对比核函数（实验核心）

a) 首先定义了一个 kernels 字典，列出了要对比的所有核函数及其对应的 libsvm 参数（-t 0 是线性核，-t 2 是 RBF 核，等等）。

b) 然后用一个 for 循环遍历这个字典：

- svm_train: 使用相同的训练数据（y_train, x_train）和不同的核函数参数（kernel_param）来训练模型。

- svm_predict: 使用刚刚训练好的模型，在相同的测试数据（y_test, x_test）上进行预测。
- accuracy = p_acc[0]: 从预测结果中提取出“准确率”。
- results[kernel_name] = accuracy: 将该核函数的准确率记录下来，以便最后统一对比。

```
kernels = {
    '线性核 (Linear, -t 0)': '-t 0 -q',
    '多项式核 (Poly, -t 1)': '-t 1 -q',
    'RBF 核 (Gaussian, -t 2)': '-t 2 -q',
    'Sigmoid 核 (-t 3)': '-t 3 -q'
}
results = {}

print("\n--- ★★★ 开始执行实验 ★★★ ---")
for kernel_name, kernel_param in kernels.items():
    print(f"\n正在训练 {kernel_name}...")

    # 1. 训练模型 (使用不同的核函数参数)
    model = svm_train(y_train, x_train, kernel_param)

    print(f"正在测试 {kernel_name}...")

    # 2. 评估模型
    p_labels, p_acc, p_vals = svm_predict(y_test, x_test, model)

    # 3. 记录准确率
    accuracy = p_acc[0]
    results[kernel_name] = accuracy
    print(f"--- {kernel_name} 的准确率 (Accuracy): {accuracy:.2f}% ---")
```

（五）实验测试结果对比与总结

为了让实验具有对比性，我利用 Pillow 分别生成了三种不同难度的验证码图片数据。

1) Hard_images:0 0 9 6

2) Medium_images:0 0 2 5

3) Perfect_images:0 2 6 2

分别在这三种不同难度的数据集下用三种不同的核函数进行训练，形成更全面的对比。

（数据量都是 1000 张图片）

结果展示：

在 perfect 和 medium 的数据集之下，线性核、多项式核、RBF 核的效果都是**极好的**（perfect 是三个 100%），medium 的结果如下：

```
Accuracy = 99.2537% (798/804) (classification)
--- 线性核 (Linear, -t 0) 的准确率 (Accuracy): 99.25% ---

正在训练 多项式核 (Poly, -t 1)...
正在测试 多项式核 (Poly, -t 1)...
Accuracy = 99.2537% (798/804) (classification)
--- 多项式核 (Poly, -t 1) 的准确率 (Accuracy): 99.25% ---

正在训练 RBF 核 (Gaussian, -t 2)...
正在测试 RBF 核 (Gaussian, -t 2)...
Accuracy = 99.8756% (803/804) (classification)
--- RBF 核 (Gaussian, -t 2) 的准确率 (Accuracy): 99.88% ---

正在训练 Sigmoid 核 (-t 3)...
正在测试 Sigmoid 核 (-t 3)...
Accuracy = 10.5721% (85/804) (classification)
--- Sigmoid 核 (-t 3) 的准确率 (Accuracy): 10.57% ---
```

由图可见，sigmoid 核的效果是极差的，就是和随机乱猜的正确是一样（十个数字猜一个：

10%）

接下来在 hard 数据集中查看对应结果：

```
Accuracy = 27.3973% (220/803) (classification)
--- 线性核 (Linear, -t 0) 的准确率 (Accuracy): 27.40% ---

正在训练 多项式核 (Poly, -t 1)...
正在测试 多项式核 (Poly, -t 1)...
Accuracy = 43.5866% (350/803) (classification)
--- 多项式核 (Poly, -t 1) 的准确率 (Accuracy): 43.59% ---

正在训练 RBF 核 (Gaussian, -t 2)...
正在测试 RBF 核 (Gaussian, -t 2)...
Accuracy = 10.7098% (86/803) (classification)
--- RBF 核 (Gaussian, -t 2) 的准确率 (Accuracy): 10.71% ---

正在训练 Sigmoid 核 (-t 3)...
正在测试 Sigmoid 核 (-t 3)...
Accuracy = 9.83811% (79/803) (classification)
--- Sigmoid 核 (-t 3) 的准确率 (Accuracy): 9.84% ---
```

可以看到，线性核和多项式核能够稍微顶住“压力”，经过训练之后的效果至少是比随机猜要好，而 RBF 则效果变得奇差，sigmoid 效果一如既往的差。

总结：从本次实验结果观察而来，我推测多项式核最适合验证码识别模型的训练。

四.实验总结

本次实验旨在利用 SVM 完成验证码识别，并重点对比不同核函数对识别效果的影响 。鉴于参考教程的数据源已失效 ，本实验改用 Pillow 库程序化生成了“完美”、“中等”和“困难”三种不同难度（包含随机噪点、旋转和干扰线）的合成数据集 。在数据处理上，实验遵循了教程的思路，采用了“固定坐标”切割法 和“行列求和”特征提取法（归

一化为 8x18，生成 26 维特征） 。在模型训练阶段，实验解决了 libsvm（旧包）与 libsvm-official（新包）的导入路径差异问题 ，并成功对比了四种核函数 。实验结果显示，在“中等”难度数据上，RBF 核（99.88%）表现最佳，而 Sigmoid 核（10.57%）几乎等同于随机猜测 。但在“困难”数据上，RBF 核的性能急剧下降，效果奇差 ，反而是线性核与多项式核能够获得比随机猜测更好的结果 。综合所有数据集的表现，本实验推测**多项式核（-t 1）**最适合用于此次验证码识别模型的训练 。